Developing a Document Trained Automated Advisor

Eric Gregori
Georgia Institute of Technology
Atlanta, Georgia, USA
egregori3@gatech.edu

ABSTRACT

This paper covers the development of a system to automatically answer questions about the content of a document. For example, a class syllabus or project specification. The system trains on the document's content to build a model and answer questions using text from the document.

Future research includes:

- advanced document parsing
- automatic question generation from text
- incremental learning (updated knowledge)
- conversational user interface

INTRODUCTION

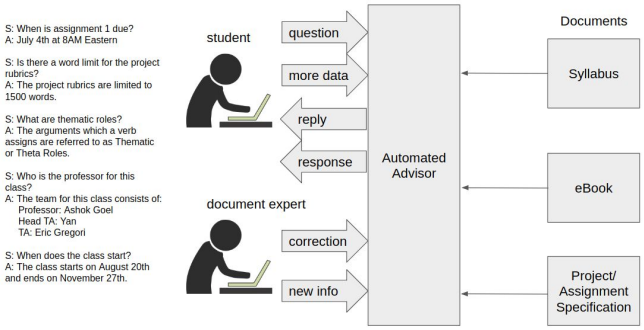

Figure 1: document trained automated advisor

An automated advisor agent, or chatbot, is a text-based human-computer interface. The automated advisor described here answers questions about the contents of a document. The system has two parts. The training portion consumes documents and trains a model on its contents. The automated advisor agent uses the model to answer the user's questions.

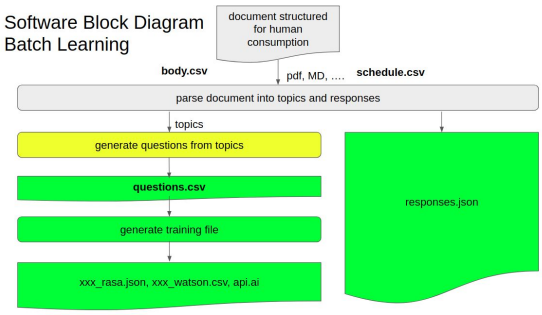

Figure 2: training on the document

Figure 2 shows the training portion of the system. The document is broken down into intents and responses. The intents are combined with questions to train a model. The model and responses are used by the agent to answer user questions.

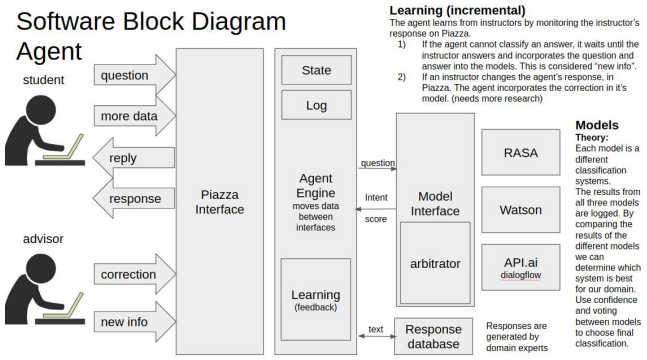

Figure 3: advisor agent

Figure 3 shows the agent execution engine. The engine executes the models trained to the document. Multiple models are trained and used for the purpose of research and robustness. An arbitration algorithm combines the classifications from the different models into a final answer.

MODULAR CONSTRUCTION

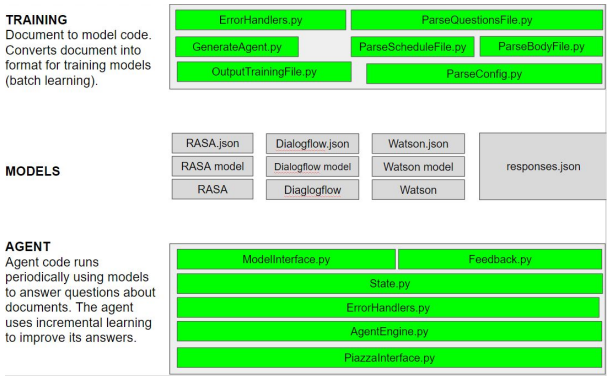

Figure 4. code construction

The code is broken up into two executables: a trainer and an agent. The trainer is used for batch learning. The agent runs on a periodic schedule answering users questions. The code is modular in construction taking advantage of Python classes for an object-oriented approach. Each module is its own class. This makes the code easy to understand and maintain. In addition to making it easy to add functionality to the agent or trainer.

BATCH LEARNING

The models encode the information from the documents during training. Each model uses a different system of information encoding. The model uses the encoded information to classify questions in text format into intents. Intents act as indexes or keys into a response database, figure 3.

Figure 2 describes the process of creating a training file from a document. The process assumes the document is human readable and has some structure. Specifically, blocks of text labeled with headers (similar to this document). It is assumed the header is the topic of the text below it (not always a safe assumption, for example: 'introduction'). The header is used to generate questions. The text block under the header is used to create the response. The system currently works with MarkDown files. Work is in progress on PDF files.

Currently, question generation is based on templates and requires significant human interaction (with the exception of schedules). There is still a lot of work that needs to be done to make question generation completely automatic.

INCREMENTAL LEARNING

Batch learning is used to create new models. Dynamically adding new information to existing models is incremental learning. If the agent cannot answer a question or answers a question incorrectly, the document expert intervenes in real-time.

If the agent does not answer a question in x amount of time, the document expert can assume the agent does not know how to answer the question. The domain expert answers the questions directly in Piazza. The agent extracts the question and answer from Piazza and adds them to the models.

If the agent answers a question incorrectly, the document expert can correct the agent by overwriting the agent's response in Piazza.

AGENT ENGINE

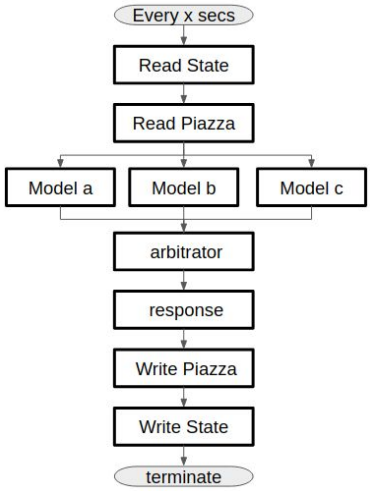

Figure 5. agent engine

The engine runs periodically as a cron job. Waking up every x minutes to scan Piazza for new questions. History is stored in a state file to avoid answering questions that have already been answered. The state file will be used by future revisions to store conversation state.

The questions are passed to the models and arbitrator. The resulting intent is used to index response from the response database. The response is sent to Piazza. The new state is saved.

The process is logged into a noSQL database. The log makes it easy to test the agent in the wild, by saving a copy of all transactions along with the meta-data. The log database can be queried while the agent is live. The log can be used to generate batch training data.

MODEL ARBITRATION

All three models are trained on the same material. They can yield different classifications due to their underlying learning algorithms. When the models yield different results, arbitration is required. The current arbitration algorithm uses the confidence score each model supplies with its classification. The model with the highest confidence wins. In the case of a tie (or close call), both results may be used to generate a response.

EXAMPLE: SCHEDULE DOCUMENT

	A	B	C	D	E	F	G
1	**Name**	**Release**	**Time**	**Due**	**Time**	**Duration(days)**	**weight**
2	Survey	Jan 8	11:59 AOE	Jan 14	at midnight URC-	6	10%
3	Assignment 1	Jan 15	11:59 AOE	Jan 21	Sunday at midnig	6	10%
4	Assignment 1 Pe	Jan 22	11:59 AOE	Jan 28	Sunday at midnig	6	10%
5	Project 1	Jan 22	11:59 AOE	Feb 11	Sunday at midnig	20	10%
6	Quarter couse su	Feb 5	11:59 AOE	Feb 11	Sunday at midnig	6	10%
7	Project 1 peer Fe	Feb 12	11:59 AOE	Feb 18	Sunday at midnig	6	10%
8	Assignment 2	Feb 12	11:59 AOE	Feb 18	Sunday at midnig	6	10%
9	Assignment 2 pe	Feb 19	11:59 AOE	Feb 25	Sunday at midnig	6	10%
10	Mid term	Feb 19	11:59 AOE	Feb 25	Sunday at midnig	6	10%
11	Mid term peer fe	Feb 26	11:59 AOE	Mar 4	Sunday at midnig	6	10%
12	Mid course surve	Mar 5	11:59 AOE	Mar 11	Sunday at midnig	6	10%
13	Project 2	Feb 26	11:59 AOE	Mar 19	Sunday at midnig	21	10%
14	Project 2 peer fe	Mar 20	11:59 AOE	Mar 26	Sunday at midnig	6	10%
15	Assignment 3	Mar 19	11:59 AOE	Mar 25	Sunday at midnig	6	10%
16	Assignment 3 pe	Mar 26	11:59 AOE	Apr 1	Sunday at midnig	6	10%
17	Project 3	Mar 26	11:59 AOE	Apr 15	Sunday at midnig	20	10%
18	Project 3 peer fe	Apr 16	11:59 AOE	Apr 22	Sunday at midnig	6	10%
19	Final Exam	Apr 16	11:59 AOE	Apr 22	Sunday at midnig	6	10%
20	Final Examl peer	Apr 23	11:59 AOE	Apr 29	Sunday at midnig	6	10%
21	End of course survey		11:59 AOE	Apr 29	Sunday at midnig		10%
22	KBAI survey		11:59 AOE	Apr 30	Sunday at midnig		10%

Figure 6. example schedule

A schedule document (figure 6) is a table of objects and fields. Each row represents an object and each column a field.

```
"when is {} due"
"at what time is {} due"
"when does {} need to be turned in"
"{} date clarification"
"what is the due date for {}"
"what time is {} due"
"when does {} need to be turned in"
```

Figure 7. template questions

Questions in the form of a template (figure 7) are used to automatically create individual questions for each object. Responses are automatically generated based on the fields. The {} in the template questions are replaced with the objects from the schedule.

USER INTERFACE

Multiple user interfaces were evaluated including web, slack, and Piazza. The web interface has the most utility. The user logs into a web page and asks a question. The web format allows for multiple responses, images, domain selection, and managed conversation.

Slack is a popular chat service with a stack structure. A user types in a text window and submits their text. Everyone logged into the channel can see the submission and respond. Each new submission is added to the bottom of the stack. The stack structure provides a natural conversation interface, with the latest entry at the bottom. The stack has no structured beginning and ending. Dialogs begin and end randomly in time with multiple conversations being held concurrently. There is no system based events to trigger to an agent that a dialog has started.

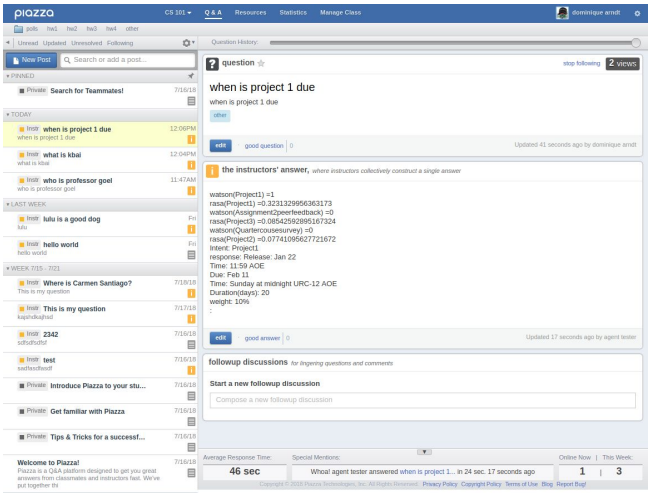

Figure 7. piazza interaction

The Piazza interface is shown in figure 3. The interface has three basic parts: subject, content, and instructors response. The agent currently looks for questions in the subject and replies in the instructor response.

FUTURE RESEARCH

The current implementation lays down the basic blocks and architecture. The next step is to expand on the functionality of each block.

Batch Learning (figure 2)

The batch learning system has two blocks: document parsing and question generation. These two blocks represent the most potential for research. The difficulty of parsing a document into topics and responses depends heavily on the author and the format of the document. A document formatted as header, text, header, text can be fairly easily parsed as long as the headers are concise and the text relates tightly to the headers. A document with few broad headers and lengthy broad text requires significantly more intelligence when parsing.

The question generation block generates questions based on the text. The current implementation uses templates or schedules but requires manual question generation for everything else. Automatically generating questions from the text is a very interesting line of research.

Incremental Learning (feedback)

The agent learns on the fly using incremental learning. The current system can add new knowledge to the models. The research potential is with updating model knowledge. When the agent answers a question incorrectly, and an expert corrects it, how is this correction incorporated in the model?

User Interface

Dyadic Conversation

The user interface is currently limited to a single adjacency pair. This is referred to as a one-shot conversation. Siri, Alexa, and Google assistant are examples of speech based one-shot conversation systems. [2]

A conversational agent can support multiple adjacency pairs. Remembering states between pairs, capable of associating data in different adjacency pairs (being able to maintain the conversation). The ability to ask clarifying question increases the utility of the agent substantially.

REFERENCES

1. Goel, A. K., Polepeddi, L. (2016). *A Virtual Teaching Assistant for Online Education.* http://hdl.handle.net/1853/59104

2. Gregori, E. (2017). *Evaluation of Modern Tools for an OMSCS Advisor Chatbot.* http://hdl.handle.net/1853/58516